Project Report

Summer 2024

CSE325 Operating System

Section: 1

Project Title: Sweet Harmony

Submitted By-

ID	Name	Roll No
2022-1-60-078	Tanjila Akter	16
2022-1-60-091	Ronjon Kar	17
2022-1-60-126	Antar Chandra Das	18

Submitted To-

Dr. Md Nawab Yousuf Ali
Professor
Department of Computer Science & Engineering

Submission Date: 22 September, 2024

Problem Statement

In the small, picturesque town of Pastelville, there is a lovely bakery called "Sweet Harmony." This bakery is famous for its scrumptious pastries and charming atmosphere. The bakery has a limited number of tables for customers to sit and enjoy their treats. To maintain the pleasant ambiance, they have a unique rule: at any given time, there can only be an equal number of customers wearing red and blue outfits inside the bakery.

Sweet Harmony's staff must manage the following aspects:

1. Allowing customers to enter the bakery while maintaining the equal number of red and blue outfits rule.

2. Ensuring customers can find a table to sit at or wait for a table to become available.

3. Managing the queue of customers waiting outside the bakery when the equal outfit rule cannot be maintained.

4. Handling customers leaving the bakery, freeing up tables, and allowing new customers to enter.

A C program can be developed using semaphores, processes, and thread mutex locks to address the above challenges:

1. Use Semaphores to manage the equal outfit rule, allowing only customers with matching outfits to enter.

2. Use Processes to represent individual customers and their actions

3. Use Thread mutex locks used to synchronize the available tables, ensuring customers can only sit at a free table.

4. Use semaphores to manage the waiting queue outside the bakery.

By utilizing semaphores, processes, and thread mutex locks, the C program can effectively simulate the charming environment at Sweet Harmony bakery, while addressing the concurrency issues and unique outfit rule that arise in this scenario.

Introduction

Using Threads, mutex locks and semaphore, I implemented a solution that synchronizes the activities of customer in bakery. This solution ensures that customers are entered the bakery by maintaining equal number of red and blue outfits rule. It also ensure that customers find a table to sit and waiting outside the bakery when equal outfit rule cannot be maintained.

Project Description

Overview

This project simulates a bakery where customers of two different types (Red and Blue) enter, find tables, enjoy pastries, and then leave. The simulation ensures that the number of customers of each type inside the bakery is balanced, preventing any type from dominating the space.

Key Features

- **Customer Types**: Customers are categorized as Red or Blue based on their ID (even IDs are Red, odd IDs are Blue).
- **Synchronization**: The project uses semaphores and mutexes to manage customer entry, table availability, and ensure balanced customer counts.
- **Concurrency**: Multiple customers are handled concurrently using POSIX threads.

Components

1. **Semaphores**:
    - queue_sem: Controls the number of customers allowed to wait in the queue.
    - tables_sem: Manages the availability of tables in the bakery.
2. **Mutex**:
    - lock: Ensures mutual exclusion when updating the counts of Red and Blue customers.
3. **Customer Function**:
    - Simulates the behavior of a customer entering the bakery, finding a table, enjoying pastries, and leaving.
4. **Main Function**:
    - Initializes semaphores and mutexes.
    - Creates and joins customer threads.
    - Cleans up resources after the simulation.

How It Works

1. **Customer Entry**: Customers wait to enter the bakery based on the availability of tables and the balance between Red and Blue customers.
2. **Finding a Table**: Once inside, customers wait for an available table.
3. **Enjoying Pastries**: Customers occupy a table for a random duration (1 to 5 seconds).
4. **Leaving**: After enjoying pastries, customers leave the bakery, updating the counts and freeing up resources.

Methods

- pthread_create( )
- pthread_join( )
- sem_init( )
- sem_wait( )
- sem_post( )
- sem_mutex( )
- sem_destroy ( )

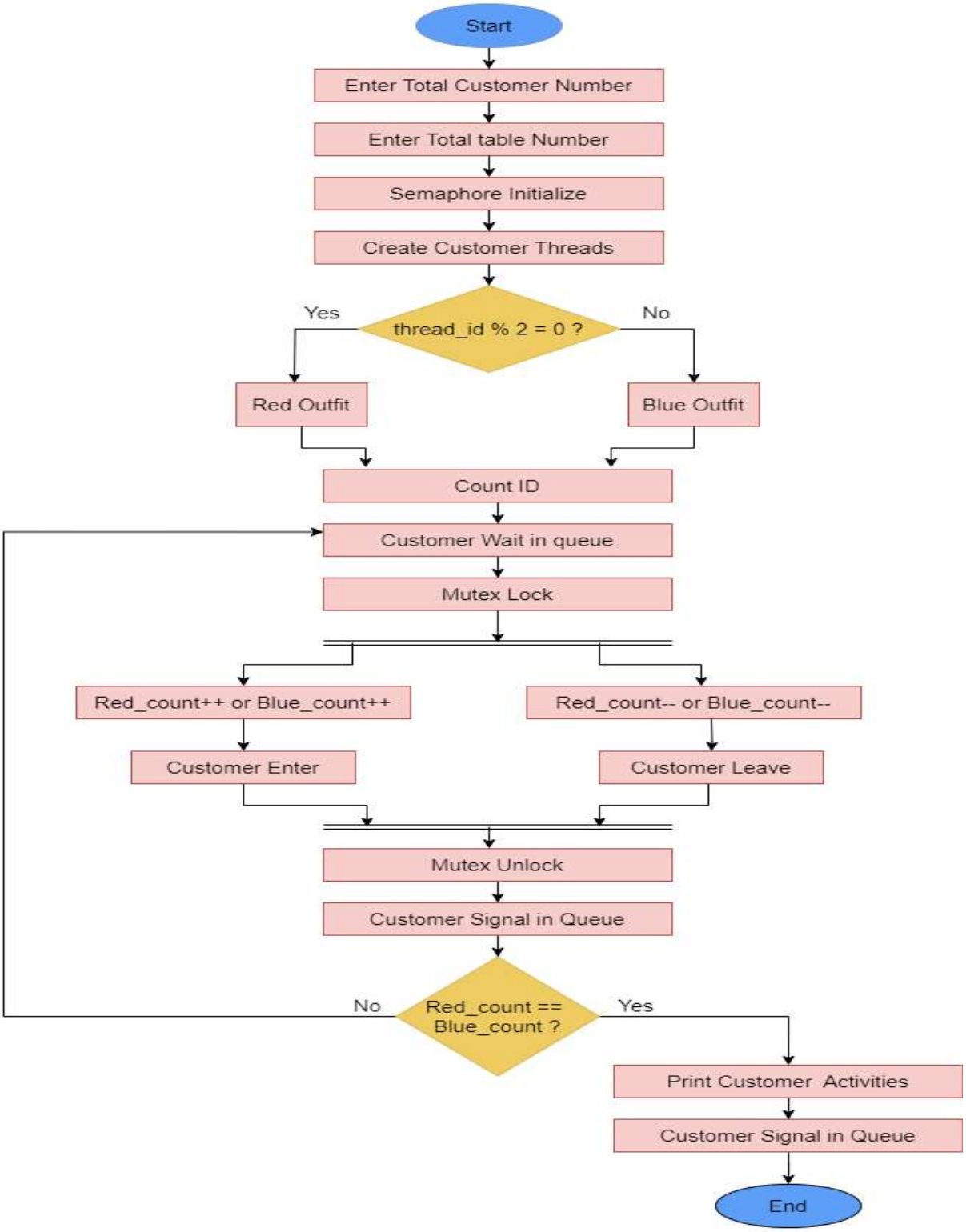
Operating System: Linux

Usage

1. Compile the code using a C compiler (gcc –pthreadsweet_harmony.c).
2. Run the executable and input the total number of customers and tables when prompted.
3. Observe the simulation output showing customer activities.

This project demonstrates the use of synchronization mechanisms in a multi-threaded environment to manage shared resources and ensure balanced access.

Process Flowchart



Code

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define max_customers 100

sem_tred_sem, blue_sem, queue_sem, tables_sem;
pthread_mutex_t lock;
int red_count = 0;
int blue_count = 0;
int total_customers, total_tables;

void *customer(void *arg) {
   int id = *(int*)arg;
   char color;
   if (id % 2 == 0) {
      color = 'R';
   } else {
      color = 'B';
   }
printf("Customer %d (%c) is waiting to enter the bakery.\n", id, color);

sem_wait(&queue_sem);
pthread_mutex_lock(&lock);
   while ((color == 'R' &&red_count>= blue_count + 1) ||
         (color == 'B' &&blue_count>= red_count + 1)) {
pthread_mutex_unlock(&lock);
usleep(100000);
pthread_mutex_lock(&lock);
   }

   if (color == 'R') {
red_count++;
   } else {
blue_count++;
   }

printf("Customer %d (%c) entered the bakery. Red: %d, Blue: %d\n", id, color, red_count,
blue_count);
pthread_mutex_unlock(&lock);

sem_wait(&tables_sem);
printf("Customer %d (%c) found a table and is enjoying pastries.\n", id, color);
```

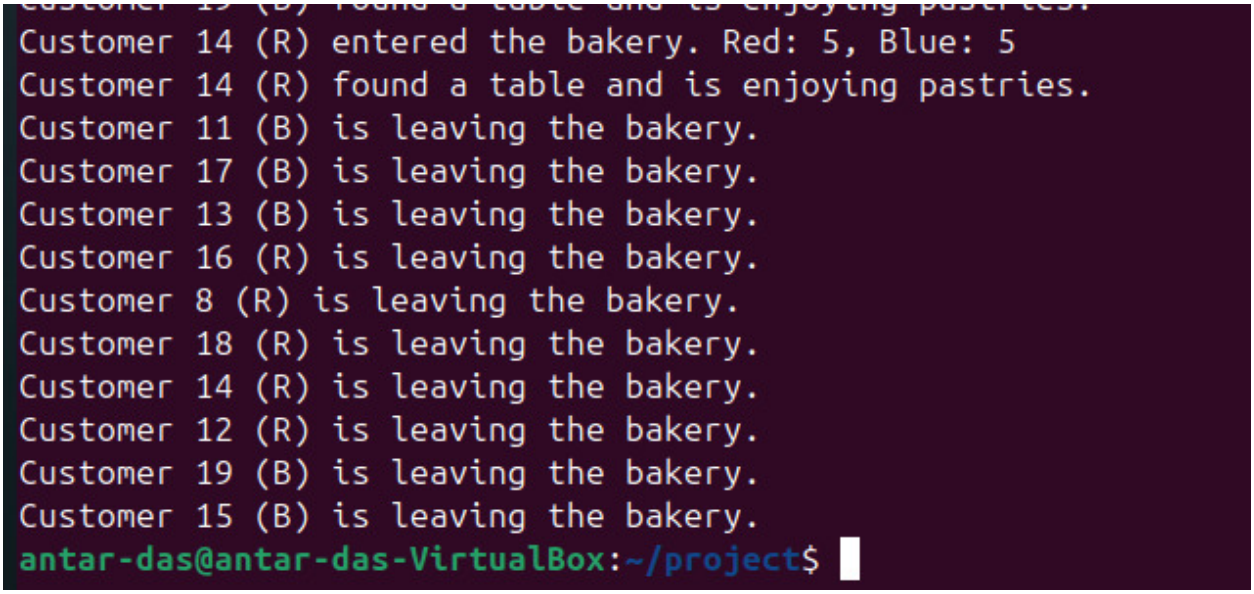
```
    sleep(rand() % 5 + 1);
sem_post(&tables_sem);
printf("Customer %d (%c) is leaving the bakery.\n", id, color);

pthread_mutex_lock(&lock);
    if (color == 'R') {
red_count--;
    } else {
blue_count--;
    }
pthread_mutex_unlock(&lock);

sem_post(&queue_sem);
    return NULL;
}

int main() {
printf("Enter the total number of customers: ");
scanf("%d", &total_customers);

printf("Enter the number of tables in the bakery: ");
scanf("%d", &total_tables);

sem_init(&queue_sem, 0, total_tables);
sem_init(&tables_sem, 0, total_tables);
pthread_mutex_init(&lock, NULL);
pthread_t threads[max_customers];
    int customer_id[max_customers];

    for (int i = 0; i<total_customers; i++) {
customer_id[i] = i;
pthread_create(&threads[i], NULL, customer, &customer_id[i]);
usleep(100000);
    }

    for (int i = 0; i<total_customers; i++) {
pthread_join(threads[i], NULL);
    }
sem_destroy(&red_sem);
sem_destroy(&blue_sem);
sem_destroy(&queue_sem);
sem_destroy(&tables_sem);
pthread_mutex_destroy(&lock);

    return 0;
}
```

Output

Let, the total number of customer is 20 and the total number of table is 10. The output is given below: -

```
antar-das@antar-das-VirtualBox:~/project$ ./a.out
Enter the total number of customers: 20
Enter the number of tables in the bakery: 10
Customer 0 (R) is waiting to enter the bakery.
Customer 0 (R) entered the bakery. Red: 1, Blue: 0
Customer 0 (R) found a table and is enjoying pastries.
Customer 1 (B) is waiting to enter the bakery.
Customer 1 (B) entered the bakery. Red: 1, Blue: 1
Customer 1 (B) found a table and is enjoying pastries.
Customer 2 (R) is waiting to enter the bakery.
Customer 2 (R) entered the bakery. Red: 2, Blue: 1
Customer 2 (R) found a table and is enjoying pastries.
Customer 3 (B) is waiting to enter the bakery.
Customer 3 (B) entered the bakery. Red: 2, Blue: 2
Customer 3 (B) found a table and is enjoying pastries.
Customer 4 (R) is waiting to enter the bakery.
Customer 4 (R) entered the bakery. Red: 3, Blue: 2
Customer 4 (R) found a table and is enjoying pastries.
Customer 5 (B) is waiting to enter the bakery.
Customer 5 (B) entered the bakery. Red: 3, Blue: 3
Customer 5 (B) found a table and is enjoying pastries.
Customer 6 (R) is waiting to enter the bakery.
Customer 6 (R) entered the bakery. Red: 4, Blue: 3
Customer 6 (R) found a table and is enjoying pastries.
Customer 7 (B) is waiting to enter the bakery.
Customer 7 (B) entered the bakery. Red: 4, Blue: 4
Customer 7 (B) found a table and is enjoying pastries.
Customer 8 (R) is waiting to enter the bakery.
Customer 8 (R) entered the bakery. Red: 5, Blue: 4
Customer 8 (R) found a table and is enjoying pastries.
Customer 9 (B) is waiting to enter the bakery.
Customer 9 (B) entered the bakery. Red: 5, Blue: 5
Customer 9 (B) found a table and is enjoying pastries.
Customer 10 (R) is waiting to enter the bakery.
Customer 11 (B) is waiting to enter the bakery.
```

```
Customer 12 (R) is waiting to enter the bakery.
Customer 3 (B) is leaving the bakery.
Customer 13 (B) is waiting to enter the bakery.
Customer 14 (R) is waiting to enter the bakery.
Customer 5 (B) is leaving the bakery.
Customer 11 (B) entered the bakery. Red: 5, Blue: 4
Customer 11 (B) found a table and is enjoying pastries.
Customer 15 (B) is waiting to enter the bakery.
Customer 16 (R) is waiting to enter the bakery.
Customer 17 (B) is waiting to enter the bakery.
Customer 18 (R) is waiting to enter the bakery.
Customer 19 (B) is waiting to enter the bakery.
Customer 1 (B) is leaving the bakery.
Customer 6 (R) is leaving the bakery.
Customer 13 (B) entered the bakery. Red: 4, Blue: 4
Customer 13 (B) found a table and is enjoying pastries.
Customer 10 (R) entered the bakery. Red: 5, Blue: 4
Customer 10 (R) found a table and is enjoying pastries.
Customer 9 (B) is leaving the bakery.
Customer 2 (R) is leaving the bakery.
Customer 15 (B) entered the bakery. Red: 4, Blue: 4
Customer 15 (B) found a table and is enjoying pastries.
Customer 12 (R) entered the bakery. Red: 5, Blue: 4
Customer 12 (R) found a table and is enjoying pastries.
Customer 10 (R) is leaving the bakery.
Customer 16 (R) entered the bakery. Red: 5, Blue: 4
Customer 16 (R) found a table and is enjoying pastries.
Customer 7 (B) is leaving the bakery.
Customer 17 (B) entered the bakery. Red: 5, Blue: 4
Customer 17 (B) found a table and is enjoying pastries.
Customer 0 (R) is leaving the bakery.
Customer 18 (R) entered the bakery. Red: 5, Blue: 4
Customer 18 (R) found a table and is enjoying pastries.
Customer 4 (R) is leaving the bakery.
Customer 19 (B) entered the bakery. Red: 4, Blue: 5
Customer 19 (B) found a table and is enjoying pastries.
```



Conclusion

In conclusion, this C program effectively simulates the charming environment of “Sweet Harmony” bakery in Pastelville by using semaphores, processes, and thread mutex locks. It ensures that the unique rule of maintaining an equal number of red and blue customers is adhered to, while also managing table availability and customer queues. By synchronizing customer actions and resource allocation, the program maintains a balanced and pleasant atmosphere inside the bakery. This simulation demonstrates the practical application of concurrency control mechanisms in a real-world scenario.